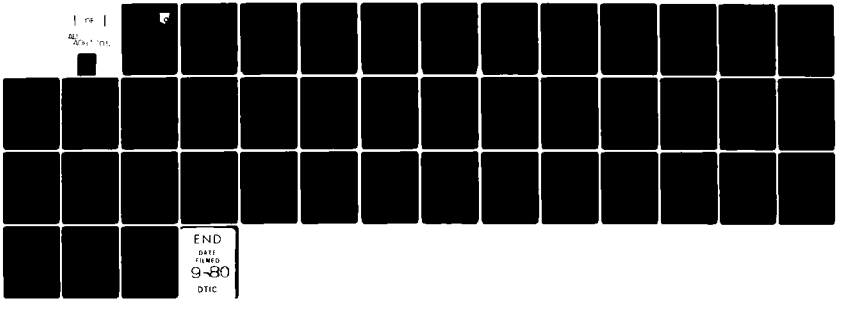
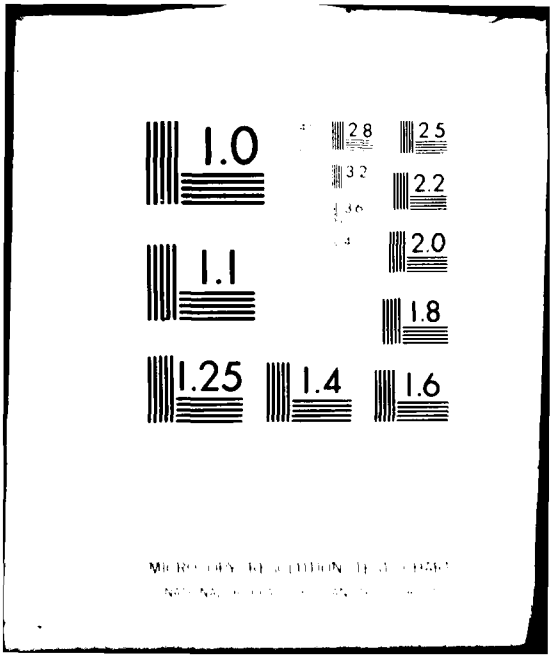
AD-A087 705 AIR FORCE WRIGHT AERONAUTICAL LABS WRIGHT-PATTERSON AFB OH F/G 9/2
ADA TEST AND EVALUATION.(U)
MAY 80 A J SCARPELLI
UNCLASSIFIED AFWAL-TR-80-1024 NL




1.0
2.8
2.5
3.2
2.2
3.6
2.0
1.1
1.8
1.25
1.4
1.6

AFWAL-TR-80-1024

LEVEL II

Ada TEST AND EVALUATION

Alfred J. Scarpelli
System Technology Branch
System Avionics Division

May 1980

TECHNICAL REPORT AFWAL-TR-80-1024

Final Report for Period 15 May 1979 - 1 January 1980

Approved for public release; distribution unlimited.

AVIONICS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report has been reviewed by the Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

Alfred J. Scarpelli

ALFRED J. SCARPELLI
Project Engineer

David J. Brazil

DAVID J. BRAZIL, CAPT, USAF
Tech Mgr, Software & Processor Grp
System Technology Branch

FOR THE COMMANDER

Raymond E. Siferd

RAYMOND E. SIFERD, COL, USAF
Chief, System Avionics Division
Avionics Laboratory

"If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify AFWAL/AAAT-2,W-PAFB, OH 45433 to help us maintain a current mailing list".

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFWAL-TR-80-1024	2. GOVT ACCESSION NO. AD-A087 705	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Ada Test and Evaluation		5. TYPE OF REPORT & PERIOD COVERED Final Technical Report 15 May 1979 - 1 January 1980
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Alfred J. Scarpelli		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Avionics Laboratory (AAAT-2) AF Wright Aeronautical Laboratories, AFSC Wright-Patterson Air Force Base, Ohio 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62204F 2003-04-22
11. CONTROLLING OFFICE NAME AND ADDRESS Avionics Laboratory (AAA) AF Wright Aeronautical Laboratories, AFSC Wright-Patterson Air Force Base, Ohio 45433		12. REPORT DATE May 1980 13. NUMBER OF PAGES 43
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Ada, Ada Test and Evaluation, Embedded Computer Systems, High Order Language Working Group, HOLWG, STEELMAN, Ada Language Environment, ALE, Ada Language Integrated Computer Environment, ALICE, Digital Avionics Information System, DAIS, JOVIAL, J73/I, Design Validation Report.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Ada is the proposed Department of Defense general purpose programming language for embedded computer system applications. The language was designed according to the STEELMAN requirements developed by HOLWG of DARPA. The primary objective of this Work Unit was to test and evaluate the Ada language in relation to the specific needs of the Air Force, as well as DoD. This effort supported the overall Test and Evaluation program sponsored by DARPA.

The AL Test and Evaluation project was to recode the Digital Avionics

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. Abstract (cont'd)

Information System (DAIS) Local Executive from JOVIAL J73/I to Ada. The results were incorporated into a Design Validation Report and submitted to HOLWG. Design Validation Reports from all Ada Test and Evaluation participants were to be evaluated and then the Ada language design would be finalized.

FOREWORD

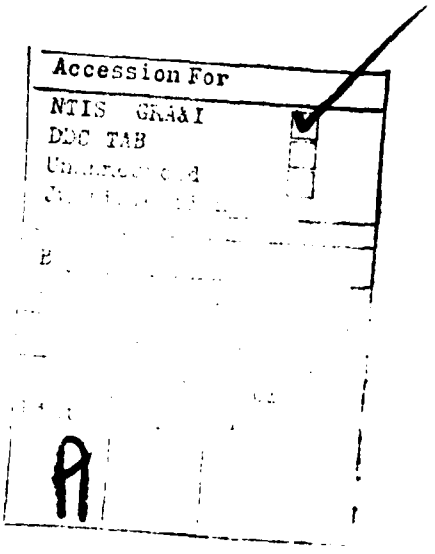
This report describes an in-house effort conducted at the Avionics Laboratory, System Avionics Division, System Technology Branch, Air Force Wright Aeronautical Laboratories, Wright-Patterson Air Force Base, Ohio 45433, under USAF Project 2003, entitled "Avionic System Design Technology," Task 04, entitled "Computer and Software Resources," and Work Unit 22, entitled "Ada Test and Evaluation."

The work reported herein was performed during the period 15 May 1979 to 1 January 1980, under the direction of the author, Alfred J. Scarpelli (AFWAL/AAAT-2), project engineer. The report was released by the author in January 1980.

The author wishes to thank Capt Steven R. Sarner, Dr. Mark T. Michael, and Guy A. Vince (AFWAL/AAAT-2), and Mike Burlakoff and Ronald Szkody (AFWAL/AAAS-1) for their contributions and assistance.

The author wishes to express his special appreciation to Capt David J. Brazil (AFWAL/AAAT-2) for his guidance, support, and assistance, and to Pamela K. Gross (AFWAL/AAAT-2) for her encouragement and moral support.

TABLE OF CONTENTS



SECTION I

INTRODUCTION

Ada is the proposed Department of Defense general purpose programming language for embedded computer system applications. An embedded computer system is integral to an electronic or electromechanical system (for example, combat weapon system, tactical system, aircraft, ship, missile, spacecraft, command, control and communication systems) from a design, procurement and operations viewpoint (Reference 1).

The program to establish a single high order programming language for use by all DoD agencies began in 1975. It was sponsored by the High Order Language Working Group (HOLWG) of the Defense Advanced Research Projects Agency (DARPA). The initial task was to define a set of requirements for such a common language. Beginning with the initial April 1975 STRAWMAN requirements, a refining process occurred, resulting in WOODENMAN, TINMAN, IRONMAN, REVISED IRONMAN, and finally, the current version, STEELMAN, dated June 1978 (Reference 2). As a parallel effort, HOLWG analyzed 23 existing languages and by January 1977, determined that none of these could meet the existing requirements (Reference 3).

Four companies, Softech, Intermetrics, Cii Honeywell Bull, and Stanford Research Institute were given contracts to develop such a language according to the STEELMAN requirements. All used the PASCAL programming language as a basis for their candidate language (Reference 3). Of the four, the GREEN language developed by Cii Honeywell Bull was chosen on 1 May 1979. The language was named Ada, in honor of Ada

Augusta, Lady Lovelace, the daughter of the poet, Lord Byron. Ada was Charles Babbage's programmer, Babbage being the inventor of the first mechanical computer.

Once the Ada language was chosen, a Test and Evaluation Program was initiated by HOLWG. The Ada language was evaluated by government, industry, and the academic sector. Each Test and Evaluation team was to recode a specific application in Ada and then relate their experiences to HOLWG through a Design Validation Report. These applications covered many varied areas in order to fully evaluate the language.

This Work Unit had several objectives. The first was to study and learn the language design of Ada. Other objectives were to determine the desirable features of Ada, and also to define the deficiencies in the existing specifications. The basis for the evaluation was to determine the suitability of Ada to the specific needs of the Air Force, as well as DoD. This Work Unit supported the overall T&E program sponsored by DARPA.

Section II is an overview of the Ada language. In Section III, the approach to the Work Unit objectives, and the events and milestones which occurred during this project, will be discussed. The primary product of this project, the Ada Test and Evaluation Design Validation Report, is presented in Section IV. Conclusions of this Work Unit are listed in Section V.

SECTION II

OVERVIEW OF THE Ada LANGUAGE

Ada is a powerful language. It is designed for use in both applications and systems programming. The language was engineered to allow reliable programs to be written which are also very readable and can be more easily maintained.

Some features of Ada include

- Strong data typing
- Algorithmic language like PASCAL or ALGOL-68
- Access types
- Support of multitasking
- Exception handling

Strong data typing forces the computer programmer to define the data type of all variables used in his application. The compiler prevents him from accidentally assigning one data type to another, or from doing unintentional mixed-mode calculations. In this method of compiler error detection, subtle programming errors can be eliminated which might otherwise take much debugging time or may not even be detected.

The language is designed to handle all algorithmic processes. Procedure flow is accomplished through such structured constructs as "if then else" clauses, "case" statements, "loop" statements, "while" clauses, "for" statements, and subprogram call capability.

Variables declared as access types in Ada have the characteristic of

being allocated dynamically. They only use storage when specifically allocated through execution of an allocator. Access types are also useful for maintaining linked list data structures.

Ada allows for the parallel execution of tasks and provides means for the rendezvous of tasks that must meet in order to pass data. Also, the capability exists for tasks to wait on other tasks which must execute prior to initial or further execution of the waiting task.

Ada allows a user to write his own exception handlers for run-time errors which may occur. Instead of relying on a system error handler, a programmer may specify code which instructs the computer on what to do should an error, or exception, arise. The language has predefined exceptions and the programmer may specify his own. These user defined exceptions can then be raised by the programmer whenever and wherever he deems appropriate.

For a detailed description of the Ada language, the reader is referred to References 4, 5, and 6.

The Department of Defense has strongly stated that there are to be no subsets or supersets of the Ada language. It is generally believed that such a situation will occur despite the DoD stand. However, DoD will not recognize any such variations as being Ada.

SECTION III

APPROACH AND MILESTONES

The Ada language design was studied during May and June 1979. The primary sources for learning Ada were the Ada Reference Manual (Reference 4), the Ada Rationale (Reference 5), and the Ada Tutorial (Reference 6). To enhance the training of Ada Test and Evaluation participants, one-week language courses were sponsored by HOLWG. These courses were taught by the language designers from Cii Honeywell Bull. The head instructor was Jean D. Ichbiah, the principal Ada language designer.

The Air Force Ada language workshop was held at the U.S. Air Force Academy, Colorado Springs, Colorado from 11-15 June 1979. Four Avionics Laboratory (AL) representatives, including the author, attended the workshop, which proved beneficial in learning Ada. This course was the first one taught, with others held at the U.S. Military Academy, the Naval Post-graduate school, Georgia Tech, and the National Physical Laboratory, Teddington, England.

For its Test and Evaluation project, AL recoded the Digital Avionics Information System (DAIS) Local Executive into Ada. DAIS is a current major project of the System Avionics Division of AL. The DAIS Local Executive, as well as all other DAIS mission software, is written in the high order language JOVIAL J73/I.

The methodology used in the recode was

- Initial definition of the executive data and data structures in Ada

- Line-for-line recode of the Local Executive routines
- Redesign the data structures and their access methods as problems were uncovered

During the recode, consultations occurred with the DAIS Program Branch to answer questions concerning DAIS and J73/I.

The AL Avionics System Analysis and Integration Laboratory (AVSAIL) DECsystem-10 computer was the primary facility used on this project. Also available were project accounts on the MULTICS systems at the Massachusetts Institute of Technology (MIT), Cambridge, Massachusetts, and at Rome Air Development Center (RADC), Griffiss Air Force Base, New York, both accessable via the ARPANET through the DEC-10. An Ada Test Translator was hosted on both MULTICS systems which was only capable of syntactic and static semantic checking of Ada source programs. It was not intended to execute Ada programs. The Test Translator saw very little use, however, since it did not work correctly. As a result, the DEC-10 was used to write and store the Ada source code of the DAIS Local Executive. MIT-MULTICS was only used to receive messages concerning the Ada T&E program and RADC-MULTICS was not used at all.

Other facilities available included the DAIS Local Executive J73/I source code, located on the DEC-10, and the DAIS library documents.

The recode phase covered the period from June 1979 to September 1979. Upon completion of the Local Executive recode, the problems which AL considered as major where determined and compiled into a presentation. This talk was given by the author at the Ada Test and Evaluation Workshop in Boston, Massachusetts on 23-26 October 1979.

The purpose of the Test and Evaluation Workshop was to allow Ada T&E participants to discuss their Ada applications, and to determine language issues which needed to be resolved. Also, the Workshop enabled those present to learn new methods and techniques for programming in Ada. The general feeling of those in attendance, including the language designers, was that the basic design of the Ada language had survived the T&E phase and that no major changes were necessary.

All results of the AL Test and Evaluation effort, both positive and negative, were gathered together into the Design Validation Report, which is presented in Section IV. This report was in the form of a questionnaire which was supplied by HOLWG. The Design Validation Reports from all T&E participants were due to HOLWG on 15 November 1979.

Evaluators could also issue Language Issue Reports (LIR). The primary function of an LIR was to call special attention to a particular Ada issue. The Design Validation Reports, along with the LIR's and input from any other sources, were all to be reviewed. Then the language would be finalized, making whatever changes were necessary.

Ada will undergo any changes determined necessary by the T&E phase during the period from December 1979 to Spring 1980. In Spring 1980, the language design will be fixed, allowing compilers and Ada support environments to be built.

An effort is currently underway to define an Ada environment. The Ada Language Environment (ALE) is the set of standards, conventions, policies, tools and procedures, and agency directives that are necessary

for effective use of Ada over the software life cycle of embedded military systems. The Ada Language Integrated Computer Environment (ALICE) is the Program Development and Maintenance Environment which aids and supports the production of programs for all applications of the Ada language -- small, medium, and large. ALICE is a key component of ALE (Reference 7). A partial list of environment tools includes loaders, linkers, text editors, debuggers, librarians, and utilities.

Definition of the requirements for the Ada Language Environment has resulted in the PEBBLEMAN series of documents. This series includes PEBBLEMAN, dated July 1978, REVISED PEBBLEMAN, and the current November 1979 PRELIMINARY STONEMAN document (Reference 8). The PRELIMINARY STONEMAN was to be refined in December 1979.

An Ada Environment Workshop was held in San Diego, California on 27-29 November 1979. The purpose was to discuss requirements and technology for the structure and content of an ALICE for production and modification of software for embedded computer applications and support (Reference 7).

SECTION IV

DESIGN VALIDATION REPORT

The Avionics Laboratory (AL) Test and Evaluation Team consisted of Alfred J. Scarpelli, Project Engineer, Capt Steven R. Sarner, USAF, AL Ada Focal Point, Dr. Mark T. Michael, and Guy A. Vince, AFWAL/AAAT-2, and Mike Burlakoff, AFWAL/AAAS-1.

The AL Test and Evaluation project was to recode the Digital Avionics Information System (DAIS) Local Executive from JOVIAL J73/I to Ada. AL would like to have redesigned the DAIS system, using Ada tasking features, but due to a lack of time and manpower, decided to recode the Local Executive as it was currently designed and determine what problems would occur in doing so.

The Local Executive was recoded and many questions arose concerning the Local Executive, Jovial, and Ada, some of which were not answered. Also, none of the Local Executive routines were run through the Ada Test Translator or the Ada Interpreter. AL relied mainly on desk-checking due to problems with the Ada Translator. As a result, the Local Executive is not debugged.

The Local Executive database was not broken down into smaller packages. The three main JOVIAL compools were translated into Ada source code as they stood, with the intent to later break down the compools once it was established which routines needed access to what data. Since time was short and the Local Executive was not debugged, this effort was not undertaken.

1. INFORMATION ABOUT THE PROBLEM YOU ARE PROGRAMMING

1.1. Give the major characteristics of the problem (e.g., mathematical computation, bit manipulation, character handling, real time processing, etc.). ATTACH DETAILED SPECIFICATION.

The Digital Avionics Information System (DAIS) Local Executive provides the system software services which are utilized by the DAIS Applications Software in each of the federated processors. These services provide for the execution of Real Time applications, sharing of common data, communication control within remote processors, and fielding of remote processor interrupts.

The DAIS hardware consists of a number of processors, AN/AYK-15's, tied to a MIL-STD-1553A multiplex data bus. Interprocessor communication occurs via a Bus Control Interface Unit (BCIU).

The Partitioning Analyzing Linking Editing Facility (Palefac) is a support tool for use by the mission programmer. One of its primary functions is to construct the Executive database for the user. The programmer need only write the mission software. Palefac will then analyze the programs and initialize all data structures in the Executive. Palefac outputs are then compiled together with the Executive routines and mission software to form a working DAIS system.

A detailed specification was enclosed with this Design Validation Report and submitted to HOLWG. These documents are official DAIS documents (Reference 9), (Reference 10).

1.2. Program use.

DAIS is an experimental development project. AL is currently able to "fly" missions in the laboratory using DAIS hardware and software.

1.3. Information about previous programming of the problem.

1.3.1. Date it was programmed.

The DAIS Local Executive, version 15 September 1978, was recoded in Ada.

1.3.2. What was your connection with the program (e.g., designer, programmer, maintenance programmer, none, etc.)?

The AL Test and Evaluation Team had no connection with the design and coding of the DAIS Local Executive. However, one member of the team, Mike Burlakoff, currently works in the DAIS Software Group and is familiar with DAIS software and the Palefac system.

1.3.3. If you are not intimately familiar with the program, do you have access to someone who is (yes/no)?

AL had access to someone who was very familiar with the DAIS Local Executive (Ronald Szkody, AFWAL/AAAS-1).

1.3.4. What was his/her relation to the program (designer/coder/etc.)?

His association with the DAIS Local Executive is as a maintenance programmer in the DAIS Software Group.

1.3.5. What language was it coded in?

The DAIS Local Executive was coded in JOVIAL J73/I.

1.3.6. What compiler or assembler version was used? For what host and target computers?

J73/I, version June 1978, was used for the application. With a few modifications, the Local Executive will compile on AL's 1 September 1979 version of J73/I.

J73/I is hosted on a Digital Equipment Corporation DECsystem10 computer.

The target computer is an AN/AYK-15, a 16-bit machine.

1.3.7. Was execution time efficiency a critical implementation constraint? How critical? Did this constraint affect the design and/or implementation approach for your example?

The Local Executive must handle its assigned tasks quickly and efficiently so that the DAIS Applications Software may run with as little overhead as possible. Thus, execution time efficiency underlaid the Local Executive design and implementation approach. In the J73/I version, use of assembly language was necessary in order to increase execution speed.

1.3.8. Will the same efficiency constraint affect your Ada design and coding approach?

The same constraint affected the Ada design and coding approach.

1.3.9. If object code and/or data space efficiency was a critical implementation constraint, how critical was it? How did this constraint affect the design and/or implementation approach?

Object code and data space efficiency are very important constraints placed upon the Local Executive. A processor has limited memory in which both the Executive and Applications Software must reside. Thus, the system programmer must insure that the Executive Software uses a minimum amount of processor memory. In the J73/I version, the JOVIAL "OVERLAY" construct was frequently employed to overlay data structures in an effort to save data space.

1.3.10. Will the same constraint on space efficiency affect your Ada design and coding approach?

The same constraint affected the Ada design and coding approach.

1.3.11. Does your example require concurrent processing? If so, is more than one processor involved? What kind of scheduling approach was used?

Concurrent processing does not occur within the software of the Local Executive. There is concurrent processing in the sense that applications programs in different processors are running at the same time. While one master executive maintains global control of the system, each processor has its own Local Executive that is responsible for scheduling resident tasks locally.

The DAIS Local Executive uses a priority scheme whereby the highest priority dispatchable task will always currently be executing. That does not mean that a low priority task which is running will be interrupted by a higher priority task which suddenly becomes capable of running. The Local Executive allows for that higher priority task to run should the lower priority task complete execution, or become suspended (e.g., a task may suspend itself by issuing a WAIT real-time statement).

1.3.12. State any other significant characteristics of the previous program that may affect your Ada design and coding approach (e.g., was reusability or portability important, was this a long life operational program to be maintained)?

The DAIS system is currently in the demonstration stage. The system is to be transferred out of the laboratory environment within the next few years. Therefore it is important that the Executive, as well as all other mission software, be easily maintainable over a long life cycle.

Transportability is always desirable, but in the case of an executive, less important than efficiency.

1.4. If you have the data for the source program as previously coded in a high level language, please give them for the previous version and later for the Ada version you develop.

The following statistics are for the programming of the DAIS Local Executive in J73/I and Ada:

	J73/I	Ada
Number of executable statements	529	533
Loop statements	18	19
Conditional statements	117	117
Procedure (not function) calls	126	127
Assignment statements	248	251
Transfers of control	20	19
Labels	1	0
Number of identifiers declared	288	285
Number of comments	775	781

1.5. If possible, submit the original program together with the Ada version.

The original program and its Ada version were included with this Design Validation Report under separate cover and submitted to HOLWG.

2. INFORMATION ABOUT YOUR APPROACH TO PROGRAMMING IN Ada

2.1. Was a significant redesign of this program done before coding in Ada (yes/no)? If so, why?

A significant redesign was not done on the DAIS Local Executive. AL started with the original Local Executive design and initially recoded all of the internal tables into Ada. The next step involved a line-for-line recode of the Local Executive routines. As problems were uncovered, a redesign of some tables, and their methods of access, occurred, involving both adding and deleting variables from the database.

2.2. In what way is the new design better or worse than the original? Why?

The redesigning of tables caused more array indexing to be performed, which will cause the Local Executive, and the DAIS system, to be less time efficient.

In the J73/I version, many cases occurred where a pointer in a table A referencing a variable in a table B actually contained a machine address offset from the starting address of table B to the referenced variable. Such a case occurs in the Minor Cycle Event Generation Tables. This type of access method was choosen to reduce access time. Table B is actually defined as an array with variable length entries. Having to calculate the array indices for this type of table each time access is necessary would involve too much overhead.

In Ada, it was not possible to access data with machine address offsets. The tables had to be defined as variable length entry arrays, using variant records with discriminants. To access these tables, many array index calculations must now take place. Unless these calculations are efficient, the Local Executive will run much slower, a situation that cannot be afforded. Even if calculated efficiently, the Ada version will still execute slower than the J73/I version due to increased array indexing.

An attempt was made at using access types to define these tables in an effort to increase the speed of the Local Executive. The attempt failed since access types cannot denote static variables. The Local Executive consists predominantly of constants allocated statically at compile time (initialized by Palefac) and the access types would have to point to these static variables.

The other alternative is to define all the tables as access types and allocate them at run-time. In section 4.10.4 herein, reasons are cited for why this solution may be unacceptable.

2.3. What Ada concepts, if any, affected your redesign?

As stated, only the data tables were redesigned. The design of the Local Executive remained constant throughout the Test and Evaluation process.

The Ada concepts that affected the redesign of the tables were variable length entry arrays (using variant records and discriminants) and access types.

2.4. How did the need for storage and/or time efficiency affect your redesign (especially if these requirements posed Ada programming difficulties later)?

Storage and time efficiency are both extremely important. In the J73/I version, much of the Executive data was overlayed in an effort to save memory. In order to increase execution speed, assembly language was used in various routines. Also, records contained pointer data that referenced other records, as opposed to referencing by array indices.

The Local Executive has tables where data exists in a record for one case and not for another (e.g., the Event Table may have entries if an event is located in another processor). In order to save memory space in the Ada version, variant records were declared so that each record of a table (or array) would use as little memory as possible.

AL is concerned as to whether the index calculations to access elements in such arrays will be computed efficiently. As mentioned in section 2.2 herein, an unsuccessful attempt was made at using access types to increase efficiency.

2.5. Did you have any problems mapping your design into Ada source code? What were they?

2.5.1.

Data arriving in messages from other processors must be queued in order to efficiently employ both the CPU and the BCIU. Hence, the data type of the reception queue must be defined. In the J73/I version, the queue consisted of eight 33 word buffers of type INTEGER. Data was removed from the queue via assembly language statements in order to save time.

It is desirable to use Ada assignment statements to remove data from the queue. To do so, the type of the queue must be identical to that of the receiving field. It was not apparent at the time of coding that UNSAFE_CONVERSION could be used to convert data in the queue to its correct type so it could then be easily transferred by Ada assignment statements.

2.5.2.

The DAIS Local Executive allows for tasks to issue certain real-time statements (e.g., READ, WRITE, SCHEDULE, CANCEL, WAIT). In the J73/I version, the real-time statements are implemented with the use of the Jovial construct "DEFINE". For example, if an applications task wishes to perform a READ operation, it can issue the statement

```
READ (INS007)
```

where INS007 is the block of data it wants to read. The READ statement is then "defined" into a call to a subroutine with INS007 as one of the parameters

```
X$ARD (INS007, D$INS007);
```

This method is used to ease the writing of software for the applications programmer, to separate Applications Software from Executive Software, and to increase readability and clarity of the Applications Software.

In Ada, this method of allowing the user to execute a real-time statement, i.e., READ (INS007) , is possible but not feasible. One implementation is to call a procedure named READ with one parameter

```
READ (INS007: in DATA_BLOCK);
```

This routine is really a dummy procedure which does nothing but call the real READ routine with all the parameters explicitly stated. The overhead is two subroutine calls for every call to a real-time statement.

AL does not recommend allowing the equivalent of a Jovial "DEFINE" in Ada. It is a useful construct for the DAIS real-time statements but once a user is given the "DEFINE" capability, he will surely overuse it and then Ada programs will lose their clarity and maintainability.

2.5.3.

When issuing a SCHEDULE real-time statement to schedule an applications task, the task name must be passed as a parameter so the Local Executive will know which Task Table B entry to access. Task Table B contains the task's starting address, and an assembly language routine is used to transfer control to the task.

Ada does not allow procedure names to be passed as parameters, so an enumeration type, TASK_NAME, was defined as

```
type TASK_NAME is (TASK1, TASK2, ..., TASKN);
TASK_NAME_TABLE: constant array (TASK1 .. TASKN) of INTEGER;
```

where TASK1 ... TASKN are the names of all applications tasks. The integers in TASK_NAME_TABLE are indices into the array TASK_TABLE_B. Both TASK_NAME and TASK_NAME_TABLE would be initialized by Palefac.

A task may now issue the real-time statement

```
SCHEDULE (TASK3: in TASK_NAME);
```

To access TASK3's Task Table B entry, use

```
TASK_TABLE_B (TASK_NAME_TABLE (TASK3))
```

In this situation, the overhead involves two array accesses every time data is needed from Task Table B.

This method of scheduling, and also the need to pass the task

name as a parameter, would not be needed if Ada tasking were used. There would be no need for a Task Table B, and calls to tasks would involve no more than issuing an "initiate" statement

```
initiate TASK3;
```

2.6. Do you feel that knowledge of Ada helped you arrive at a better design? If so, how?

No, it did not. The DAIS Local Executive was recoded as it was previously designed with only minor changes. AL does feel that the Local Executive could have been redesigned and improved by making use of the Ada tasking features. However, time did not permit a deeper investigation into the suitability of Ada tasking to the project.

2.7. Did you change your design as a result of trying to program it in Ada? Why? (This question should be answered each time Ada coding difficulties or advantages made you decide to change your design or design approach.)

The basic design of the Local Executive remained constant throughout the Test and Evaluation process. As previously mentioned, only the table definitions, not their basic content or purpose, changed.

2.8. Did you develop your program by yourself or as a team? If as a team, give the account identifiers of the team members that influenced the design or coding approach.

The majority of the effort was done on my own. When problems arose concerning DAIS, J73/I, and/or Ada, the other members of the team were consulted. Either solutions were found, or the problems were documented for insertion into this report.

The AL Test and Evaluation Team consisted of:

	ARPANET	MIT-Multics
Alfred J. Scarpelli	SCARPELLI@AVSAIL	Scarpelli.AdaTE
Capt Steven R. Sarner, USAF	SARNER@AVSAIL	SSarner.AdaTE
Dr. Mark T. Michael	MICHAEL@AVSAIL	MMichael.AdaTE
Guy A. Vince	VINCE@AVSAIL	**********
Mike Burlakoff	**********	Burlakoff.AdaTE

3. SPECIFIC DIFFICULTIES OR CONCERNS ABOUT LANGUAGE FEATURES OR RESTRICTIONS OF Ada

The following questions deal with the features of Ada. A feature may be general (e.g., exception handling) or specific (e.g., underscores in numeric literals). In answering the questions, please identify what features you are discussing in terms of sections of the Ada Reference Manual (Reference 4) documentation.

3.1. Which language features did you find difficult to learn to use correctly? Briefly describe the problems (e.g., Reference Manual not clear) and their severity.

3.1.1.

Record Types (section 3.7, Reference Manual) - It was not clear if defining an array of variant record types was legal. The construct of variant parts seemed to allow array elements to be of variable length. It also seemed possible to define variable-length arrays inside of variable-length arrays. After determining that the above are legal, accessing these complex data structures efficiently became a major concern.

3.1.2.

Access Types (section 3.8, Reference Manual) - The concept of access types was not described in much detail in the Reference Manual. The Rationale was clearer in its description; but useful, simple examples were not given in either.

3.1.3.

Restricted Program Units (section 8.3, Reference Manual) - Use of the restricted clause and what it accomplishes is not easily determinable. In nested procedures, the restricted clause restricts access to some of the outer procedures, depending on its use. However, use of the restricted clause to reference procedures at the same declaration level as the procedure issuing

the restricted clause achieves visibility to those other modules. In fact, there is an implicit restricted clause in front of every procedure which gains the visibility needed to call other procedures. It had been stated that the restricted clause never increases the visibility of a procedure, yet in this case it does.

3.1.4.

Tasks (section 9, Reference Manual) - The tasking of Ada may have been an improved alternative to the way DAIS is currently designed. However, due to a lack of time and manpower, and the complexity of tasking, AL was unable to apply tasking to the DAIS system.

3.1.5.

Unsafe type conversions (section 13.10, Reference Manual) - The Reference Manual and the Rationale mention very little on UNSAFE_PROGRAMMING. It is not stated whether the use of UNSAFE_CONVERSION actually generates object code and does a conversion of the data. Nor is it explicit that use of this generic package is meant to view data in different representations.

In the DAIS Local Executive, messages arriving over the data bus had to be queued. Since the data can take on many formats, it was not evident how to use Ada assignment statements to move data in and out of the queues. It was pointed out at the Ada Test and Evaluation Workshop in Boston on 23-26 October 1979 that UNSAFE_PROGRAMMING was the answer.

3.1.6.

ADDRESS attribute (Appendix A, Reference Manual) - In the AN/AYK-15, 16-bit words yield precision from -32768 to 32767 but actual memory addresses range from 0 to 65535. In another case, if virtual addressing is used, addresses will be greater than the precision of the machine can hold. What the ADDRESS attribute yields in such cases is not understood.

3.1.7.

In many instances, the Ada Reference Manual is not clear on the topic it is trying to describe. The sections which were not clear

usually lacked sufficient documentation and useful examples. In many cases, one good example would have enlightened the reader about the subject. However, in most sections of the Reference Manual, examples were few, taken out of context, and not too detailed. Future versions should include explicit, as well as additional examples, and also increased documentation.

3.2. Which features did you find difficult to apply (even after having learned how to use then correctly)? Describe the problems.

3.2.1.

AL could not apply access types (section 3.8, Reference Manual) to its project. The Local Executive tables consist predominantly of constants with various relationships between them. Access types could have been used to greatly improve access performance, but they are not capable of denoting static variables.

To use access types, all the constants and tables would have to be declared as access variables. AL did not attempt to define all Local Executive tables as access types. At the time of recoding the tables from J73/I to Ada, not enough was known about access types to do so. It is now clearer that this approach could have been used effectively. However, the access types would have to be initialized at run-time, a process with possibly too much overhead (see section 4.10.4 herein).

3.2.2.

Interprocessor communication in the DAIS system is done over a multiplex bus system. When one processor wishes to send a message to another processor, it places the message in its transmission queue. The BCIU reads that data from memory and sends it over the bus to the correct processor independent of the CPU. The receiving processor's BCIU stores that message in a memory buffer specified by the contents of a user programmed, fixed memory location. It is desirable to allocate reception buffers dynamically, and only of the required length, since messages can be of variable length.

AL attempted to create a message reception queue by using access types (section 3.8, Reference Manual) to form a dynamic linked list of incoming messages. Assuming that the word count is contained in the first word of the message, a type Q_BUFFER is defined as follows:

```
type Q_BUFFER is access
   record
      WORD_COUNT: constant INTEGER;
      MSG: array (1 .. WORD_COUNT) of INTEGER;
      Q_LINK: Q_BUFFER;
   end record;
```

Before a message arrives, the BCIU must know the address where the message should be placed. The following object declaration is issued:

```
BUF1:= new Q_BUFFER (???, ???, Q_LINK => null);
MSG_BUF_ADDR:= BUF1'ADDRESS;
```

Withholding discussion of the field qualifications of BUF1 for later, when the message arrives, the BCIU will store it at the memory address contained in MSG_BUF_ADDR and a message arrived interrupt will occur. The following code would then be executed in the associated interrupt handler:

```
BUF2:= new Q_BUFFER (???, ???, Q_LINK => null);
BUF1.Q_LINK:= BUF2;
MSG_BUF_ADDR:= BUF2'ADDRESS;
```

The new buffer, BUF2, is allocated, the link from BUF1 to BUF2 is established, and the BCIU is informed of the address of BUF2.

The new buffer must be allocated before the message arrives since the BCIU, which operates independently of the CPU, must know where to store the message. To allocate the buffer, all fields of type Q_BUFFER must be qualified. To do so, the WORD_COUNT must be known. However, the WORD_COUNT will not be known until the message arrives, at which time it is too late to allocate the buffer.

The message buffer cannot be allocated at the time the message begins to arrive because the BCIU only generates an interrupt when the message is completely received, not at the start of the message.

For these reasons, the attempt using access types failed.

3.2.3.

The use of UNSAFE_PROGRAMMING (section 13.10, Reference Manual) to handle the transmission and reception queues was not attempted due to not understanding the significance of this package until the Ada Workshop. However, it is of concern as to whether the use of UNSAFE_CONVERSION actually generates executable code. Does a real conversion actually take place? If so, the overhead of function calls to UNSAFE_CONVERSION, and execution of its code, could be prohibitive.

3.3. What language features seem to be missing in Ada that you needed for your application? (Provide an answer even if you later discovered a way of using Ada to meet your needs.)

3.3.1.

A method of representing data in different formats. It was later discovered that UNSAFE_PROGRAMMING (section 13.10, Reference Manual) is the method meant to handle such requirements.

3.3.2.

An efficient method of implementing the DAIS real-time statements is necessary (see sections 2.5.2 and 2.5.3 herein).

3.3.3.

The FOR loop construct (section 5.6, Reference Manual) is only capable of incrementing or decrementing the loop by one each time through the loop. There are situations in which a programmer may wish to loop by values other than one (2, 3, 8, 200, -10, etc.). It is desirable to allow the user to increment and decrement the FOR loop by any value he wishes.

3.3.4.

The DAIS Local Executive has control over the states of tasks (Instruction Counter (IC), general registers, condition status, and workspace pointer). It handles the saving of the processor state when an interrupt occurs, and restores the state upon return from interrupt. Ada, or any other HOL including JOVIAL, does not give access to this low level environment. Thus, assembly language was used in both JOVIAL and Ada versions to access and control the states of tasks.

The DAIS approach allows for an interrupted task to be suspended and another task of higher priority to begin execution. When the higher priority task completes, or is itself interrupted, it is not guaranteed that the task just previously suspended will immediately resume execution. Hence, a simple pushdown stack for allocating task working space for user data, compiler temporaries, etc., is not a workable solution. A task that was suspended and then restarted may expand its workspace on the stack, overwriting a suspended task's workspace. Therefore, the Local Executive must be able to manage the workspace pointer in order to tell the compiler which data space to use when the Local Executive determines the next task to run.

JOVIAL allows based data. To implement the DAIS approach, a stack area is allocated for each task and the Local Executive is informed by Palefac at compile time where the data space for each task is located. When an interrupt occurs, the IC, general registers, condition status, and current location of that task's workspace pointer are all saved (by assembly language) in the entry designated for that task in Task Table B.

Having this data stored in Task Table B and maintaining a workspace area is what allows the Local Executive to execute any task next. Task workspace areas are not stacked. Rather the maximum amount of storage needed by the task is computed by Palefac and allocated at compile time.

AL could not determine how to implement this approach in Ada. Although assembly language could be used to load and store the IC, general registers, and condition status, the Local Executive still needs control of the workspace pointer.

3.3.5.

DAIS operates on a time period called Minor Cycles. There are 128 Minor Cycles per second, constituting one Major Frame. Tasks are scheduled to run on certain Minor Cycles in each Major Frame, starting on Minor Cycle X (the phase) and running every Y Minor

Cycles (the period). For example, a task monitoring sensor input may have to run on Minor Cycles 2, 6, 10, 14, 18, etc. The phase is 2 and the period is 4.

If Ada tasking is to be used, there must be a capability to initiate a task every X microseconds if it is necessary that a task run at such a period.

3.4. Were there any interactions between Ada features and restrictions that caused you difficulties? Describe the problem.

All problems encountered with Ada have been documented in other sections of this report. The majority of the problems concern access types.

3.5. Please describe the difficult choices you encountered in developing your Ada examples and what you felt was the right choice.

In the Minor Cycle Event Generation Tables, three arrays, including the Event Table, had to be defined. Since these three tables are accessed at least 128 times a second by one procedure alone, it is necessary to have efficient access to these tables. AL wanted to use access types (section 3.8, Reference Manual) to define two of the three tables (MCT1 and MCT2) in order to increase access time efficiency. This use was not possible since access types are not able to denote static variables (i.e., access type MCT2 would have to point to the statically defined Event Table).

In the J73/I version, MCT1 contained machine address offsets pointing to MCT2 and MCT2 also contained offsets, addressing the Event Table. All three tables were defined as arrays but these offsets eliminated the array indexing that would have to occur in order to access the data in the tables.

As a result, AL decided to use arrays to define MCT1 and MCT2. The decision was not a favorable one as added array index calculations, especially on variable length entry arrays (e.g., the Event Table), will clearly increase access time and lower Executive efficiency.

3.6. Was it possible to express your program clearly and yet in a way that seems to permit a good compiler to generate efficient code? Or are you concerned that certain Ada constructs in your program may be compiled inefficiently?

The DAIS Local Executive data structures contain some arrays where the elements of those arrays are of variable length. Variant records (section 3.7.2, Reference Manual) were used to map this design into Ada.

AL is concerned about variable length element arrays. The concept is good as long as the compiler is able to generate efficient code to access these elements. Since the lengths of the elements are different, calculation of indices are not as simple as with arrays containing elements of constant size. The Local Executive is dealing in a time-critical environment, so efficient code to access data structures is a necessity.

4. OVERALL EVALUATION

4.1. After allowing for familiarization with Ada, and based on your Ada coding experiences, do you think developing a debugged program similiar to the one you did might take longer for you in Ada than in some other HOL or assembler?

> In most cases, it should not take longer to develop a debugged program in Ada once Ada is learned to be used correctly and Ada programming techniques are formed. The language is well structured, and the strong typing can prevent many errors at compile time which in another language may take a long period of time to detect, depending on how subtle the errors are.

4.2. Ada's strong type checking requires greater specification of data and its usage than is customary in many other languages. Was this strong typing an important feature that helped to detect programming errors? What errors detected?

> Detection at compile time did not help, since desk checking was entirely relied upon. The strong typing did play a vital role in the recode. The type definition of all data was closely monitored to prevent illegal assignments. Also, use of enumeration types allowed specification of all values that a particular variable could have, making the program more reliable.

4.3. In examining your Ada program (and other examples), do you believe the code is more readable than code you have previously seen? In what way, and why?

4.3.1.

> Ada source code appears much more readable than code generated in other high order languages. The following are some general comments:

- Using upper case for variables and lower case for Ada reserved words make the Ada constructs stand out better (e.g., if then else clauses).

- Double "-"s (--) to indicate comments make the comments stand apart from the code.

- Longer identifiers (or variable names), and the use of underscores to separate words within the names (e.g., TASK_TABLE_B) make them easier to read, with a fuller understanding of what they mean.

- Structured programming, including indentation of the structured constructs and matching "end" statements, make program flow easy to follow.

- The use of "others" and "null" make programmer intent very clear.

- The ".all" convention clearly states the situation, as well as making programming much easier. It eliminates having to repeat source code for all members of the structure it is referencing, making the program easier to read and maintain.

4.3.2.

AL suggests the following for improving the readability of Ada:

- The "space" and "skip" directives should be added to compiler. "Space X" would leave X blank lines in the source listing. "Skip Y" would skip down 1/3 or 1/2 of the page (or some other increment). Their use would space out comments and code, making the source easier to read.

- Use a CALL or PERFORM statement in procedure calls. Stating only the procedure name, and any parameters, does not give the reader a clear picture of what the program is actually doing (especially if there are no parameters).

- The "restricted" clause is a misleading statement. In some cases it actually increases visibility, allowing two procedures declared at the same level to reference each other. With nested procedures, its use is clearer but still confusing. AL recommends the construct "restricted to", which shows more clearly where access is restricted to.

- The syntax of the "case" statement must be changed. "case X of when =>..." does not sound grammatically correct. AL suggests removing the "of", leaving "case X when =>...".

This construct is easier to read yet still maintains the meaning intended.

4.4. Based on your experience in using Ada, list some advantages that would probably accrue to a project in your application area if it used Ada exclusively.

The Ada source code would be self-documenting and much easier to read and follow. A maintenance programmer would have a greater understanding of the program with less effort.

Ada promotes top-down structured programming.

The job of transporting programs from one machine to another would be easier.

Strong typing would prevent subtle type errors and maintain the integrity of the data. Proper use of packages will prevent procedures from accessing and modifying data that they should not have access to.

4.5. Describe the problems the project in your application area might encounter if it used Ada exclusively.

Much manipulation occurs with data that arrives over the data bus and is placed in queues. The data must be viewed in several different formats. The use of UNSAFE_CONVERSION could be a severe problem if the function actually generates machine code which must be executed every time a conversion is needed.

The DAIS Local Executive would have to use assembly language to manage the workspace pointer and to access the Instruction Counter, general registers, and condition status (see section 3.3.4 herein).

The size of the Ada run-time support and the software compiled by the Ada compiler is of great concern when dealing with machines of limited memory space. Inefficient compilation and a large run-time package may generate code that will not fit into memory.

There is concern for possible inefficient accessing methods which will slow down the system (i.e., accessing arrays with variable length entries).

The overhead in run-time initialization must be considered if

access types are to be used (see section 4.10.4 herein).

4.6. How would you feel about doing your next embedded computer project in Ada, and why?

Ada is a very capable language. It permits good structured code and the strong typing helps maintain data integrity. AL would like to use Ada in future projects. However, inefficiencies in the language may force the continued use of JOVIAL for real time applications unless the problems are resolved.

The greatest concern is with access types. The basic concept is excellent but the associated problems make them of limited value for AL applications.

4.7. What features do you feel are redundant, i.e., could be deleted without impairing the usability of the language because an alternate method of meeting a programming need exists?

No redundant features were uncovered.

4.8. What are the five most important changes you think should be made to Ada? Why?

Allow compile time initializations of access types, as opposed to doing initializations at run-time (see section 4.10.4 herein).

There is too much overhead in forcing all subprograms to be automatically recursive and reentrant (section 6.2, Reference Manual). Reserved words such as "RECURSIVE" and "REENTRANT" (used in J73/I) should be used to identify to the compiler a recursive or reentrant subprogram.

Force all procedure calls to use a reserved word "PERFORM" before the procedure name to improve readability.

Allow for read only packages. Data contained in packages could be protected against procedures which only read the data but should not modify it.

Implement the "restricted to" clause to improve readability. Also, force a programmer to state explicitly at the beginning of a

procedure, all procedures that its visibility is restricted to (i.e., eliminate implicit restricted clause), once again for clarity.

4.9. What aspects of Ada did you particularly like and would not want to have changed? Why?

Strong typing - The strong type checking eliminates errors and improves program reliability.

Packages - The partitioning of data into packages for use by selected routines will maintain data integrity. Also, the ability to define procedures inside of packages, and hiding the procedure bodies from users if desired, is a good concept. Thus, libraries of routines can be defined, and their inner workings kept secret.

4.10. Any other comments not covered above.

4.10.1.

The JOVIAL language offers a tracing feature to aid in the debugging of programs. It is recommended that such a feature be added to the Ada Environment.

4.10.2.

Range constraints on types and subtypes are vital for maintaining the integrity of the data. The construct prevents types from taking on illegal values. However, the overhead of all these run-time checks could be too great in a time critical application. Thus, to avoid this added delay in time, a programmer may leave out the range checks, losing an important language feature. Range constraints are a good idea but may not see much use in real time applications where fast execution is essential.

4.10.3.

In some applications, many records less than the length of one memory word can be tightly packed into a small number of words. The data usually contains some logical relationship that access types could express quite easily. However, to use access types, an extra word must be added for each one of these records to maintain the link, and the relationship, between the records. This word would contain the pointer to (or address of) the next member in the link. Thus, if six objects are packed into one word, at least seven words are now necessary to represent this data structure of access types. Too much memory is used if access types are employed, so the implementation of access types is not feasible for this application even though the concept of access types is.

4.10.4.

To use access types, the access variable must be allocated by use of the allocator "new" at run-time. It would be extremely desirable to allocate access types at compile time. The DAIS Local Executive has tables which have relationships that could be expressed by access types. Also, the use of access types would greatly improve access time efficiency. Much of the data

contained in the tables is known at compile time (initialized by Palefac). However, the current language design forces these tables to be allocated at run-time if access types are used. The overhead associated with run-time allocation can be excessive.

Suppose memory costs a dollar a bit (an accurate figure for some systems), and suppose it takes 480 extra bits per computer to do these run-time initializations. The cost is $480 per computer. If the government purchases only 1000 of these computers, and associated Ada software, the cost is $480,000 extra dollars for an initialization process that could have occurred at compile time. While it is true that $480,000 may only be a small percentage of the total cost of the project, over the course of many projects using Ada, the cumulative dollar amount of these small percentages will soar. Thus, a tremendous problem with access types exists.

Allowing the linking loader to do the access type allocations defeats the language. The language must allow for this feature. Otherwise, with memory space at a premium, and such high overhead, access types will certainly not be used.

SECTION V

CONCLUSIONS

In accomplishing the objectives of this Work Unit, several conclusions were drawn concerning the work done and the Ada language. As stated in the Design Validation Report, the DAIS Local Executive was recoded as it was previously designed, with minimal changes. This approach was suggested by HOLWG. As also previously stated, AL feels that the DAIS Local Executive could have been redesigned and recoded more efficiently using the Ada tasking features.

Conclusions regarding the language itself were largely presented in Section IV. Ada was a capable language for implemention of the current DAIS Local Executive approach as far as coding is concerned. AL feels that Ada is a good, strong language with many useful features. The language can be used to program any algorithm. Regarding execution speed and efficiency, it is difficult to determine how Ada would handle the real-time execution requirement without actually having an Ada compiler available. Ada's ability to perform would depend on how efficient the Ada compiler is in generating code for the target machine. The true test of the language will come when such facilities are available in 1983.

REFERENCES

1. Barry C. DeRoze, An Introspective Analysis of DOD Weapon System Software Management, Defense Management Journal, October 1975, pp. 2-7.

2. Department of Defense Requirements for High Order Computer Programming Languages "STEELMAN", DOD High Order Language Working Group, DARPA, June 1978.

3. Dr. Edward Lieblein and Edith W. Martin, Military Computer Family, Part V: Software for Embedded Computers, Military Electronics/ Countermeasures, July 1979, pp. 52-54.

4. Jean D. Ichbiah et al., Preliminary Ada Reference Manual, ACM SIGPLAN Notices, Vol. 14, No. 6, Part A, June 1979.

5. Jean D. Ichbiah et al., Rationale for the Design of the Ada Programming Language, ACM SIGPLAN Notices, Vol. 14, No. 6, Part B, June 1979.

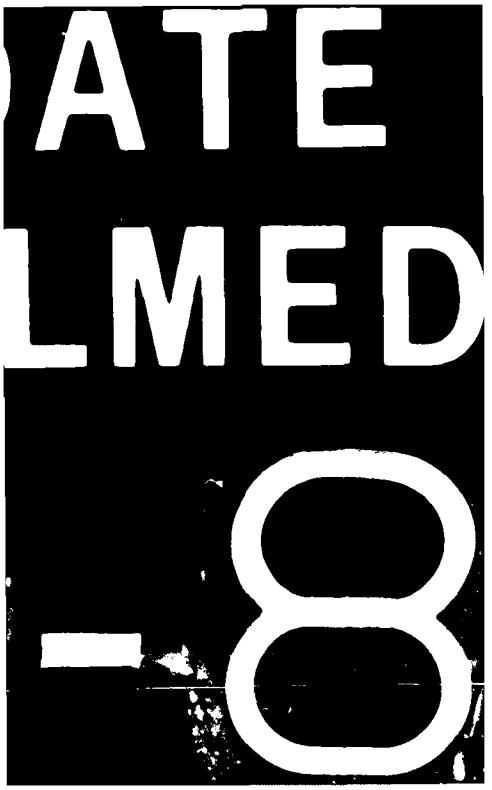
6. Jean D. Ichbiah et al., A Tutorial: An Informal Introduction To Ada, Cii Honeywell Bull, Louveciennes, France, April 1979.

7. Proceedings of the Ada Environment Workshop (San Diego, California, 27-29 November 1979), DOD High Order Language Working Group, DARPA, November 1979.

8. Department of Defense Requirements for Ada Language Integrated Computer Environments "PRELIMINARY STONEMAN", DOD High Order Language Working Group, DARPA, November 1979.

9. DAIS Mission Software Product Specification, Executive, Vol. 1: Local Executive, Air Force Avionics Laboratory, Wright-Patterson Air Force Base, Ohio 45433, SA 201302 Pt 2, Vol. 1, 27 August 1976.

10. Computer Program Design Specification for DAIS Mission Software Executive, Air Force Avionics Laboratory, Wright-Patterson Air Force Base, Ohio 45433, SA 201302A, 1 June 1979.


ATE
LMED
-8